DISTRIBUTED AND PARALLEL COMPUTING IN *MADM* DOMAIN USING THE *OPTCHOICE* SOFTWARE

CORNEL RESTEANU[1], MARIUS SOMODI[2], MARIN ANDREICA[3] and ELECTRA MITAN[1]

[1] National Institute for Research and Development in Informatics,
8-10 Averescu Avenue, 011455, Bucharest 1, Romania

[2] Department of Mathematics, University of Northern Iowa
Cedar Falls, IA 50614, USA

[3] Economic Studies Academy,
6 Romana Square, 010572, Bucharest 1, Romania

Abstract: - The paper presents a method for solving the general Multi-Attribute Decision Making (*MADM*) problems, by distributed and parallel computing, with the *OPTCHOICE* software. One presents the scheduling and load balancing algorithm for concurrent solving of sets of such problems on a given number of parallel computers. An analysis on the construction of a *MADM* problem is made; in this way, a decomposition tree having the decision-makers on the first level, the states of nature on the second level, and the attributes of the problem on the third level is emphasized. Corroborated with the analysis of the problem's data, the above results conduct at the conviction that a parallel algorithm for solving the *MADM* general problem, starting from the *MADM* particular problem, is possible. At each tree's level one can state independent particular sub-problems that are solved in parallel, the sub-problems at a superior level waiting for the solutions of the sub-problems at the current level. Finally, the classical TOPSIS method is presented running in the parallel and multi-level context.

Key-Words: - High Performance Computing, Parallel Computing, Load Balancing Algorithms, WEB Enabled Optimization, Pervasive Software, Multi-Attribute Decision Making.

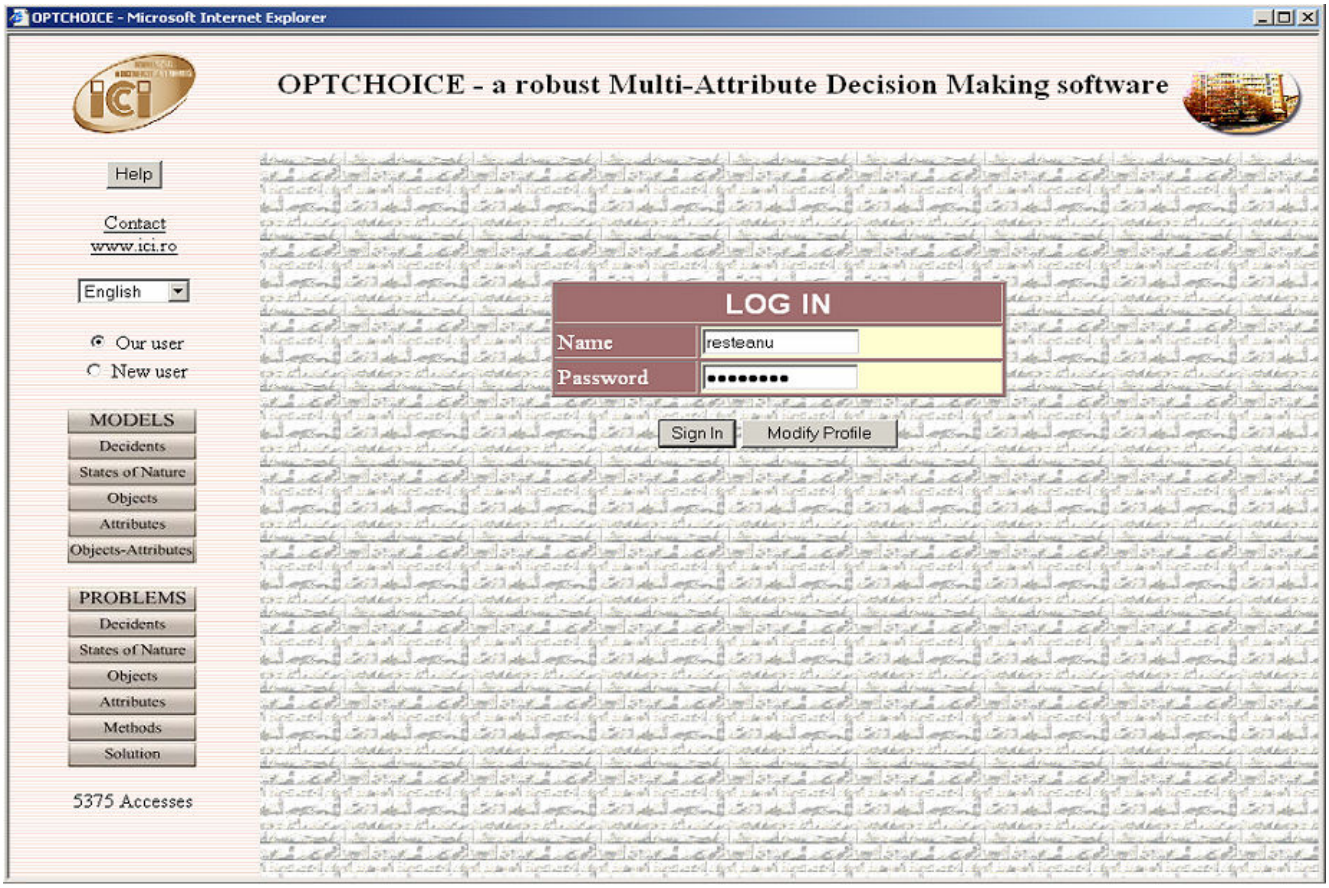
1 Introduction

Nowadays, the Internet is undergoing a fast development of services. *Web enabled optimization* is a new trend in treating Operations Research (OR) problems over the Internet [1], [2], [3], [4], [5]. Part of this new trend, the *OPTCHOICE* software is one of the first Internet-based programs designed to describe *Multi-Attribute Decision Making* (MADM) [6] [7] mathematical models, define *Optimal Choice Problems* (OCPs) [8] on them, and solve these problems in informatics performance conditions [9]. One can say that any formulation of a problem that involves choosing an object from a discrete objects set, according to an algorithmic procedure, leads naturally to a MADM model. This is one of the reasons this area's software has always been keeping pace with the progress registered in informatics. This section presents the *OPTCHOICE* software, which may be characterized as a pervasive optimization service. Recall that an Internet service is *pervasive* if it is available *to any client, free of charge, anywhere, anytime and without delay* [10].

The MADM mathematical models used by *OPTCHOICE* are general, in conformity with the decision science practices. In these models, the main entities are represented by decision makers, states of nature, objects, and attributes, whereas the link entities are represented by objects' characteristics, importance of main entities and a set of production rules for unstructured information acquisition and processing. The attributes are also general, being of cardinal, ordinal, Boolean, fuzzy or random variable type. It is common to impose the additional condition that the attributes are mutually independent variables. These mathematical models benefit from knowledge-based computing [11] to avoid their inconsistencies (syntactically and semantically incorrect / incredible / incomplete model definition). Defining an OCP in *OPTCHOICE* involves establishing its maximal range in rapport with the model description and specifying the solving methods to be used. The goal of an OCP is to select an object such that the attributes under consideration to be satisfied in an optimal way. This problem can have a high level of complexity even when only a few attributes are considered. For a large number of attributes, the complexity of the problem increases significantly. This is a consequence of the fact that, in most real-life situa-

tions, the attributes are conflicting, in the sense that one object can rank high with respect to one of the attributes but low with respect to another attribute.

There are essentially two classes of OCP solving methods: first, methods that produce explicit object evaluations (by using a set of such methods, one associate to each object an evaluation vector) and second, methods that produce object characteristics (these analysis methods associate to each object a matrix of discriminators). *OPTCHOICE* implements ten methods from the first class, namely the *maximax, maximin, non-dominance, linear utility function, scores, diameters, Onicescu, Pareto, TOPSIS, TODIM* methods, in conjunction with several normalization methods, and six methods from the second class, namely *methods of dominance analysis*. Since each evaluation method reflects a different point of view about optimality, it is clear that applying different methods to the same set of data will often lead to different solutions. In this way, an additional type of inconsistency can occur, i.e. multiple solutions, which may lead to a decisional dilemma. An inferential procedure implemented in *OPTCHOICE* addresses this drawback; it proposes a global solution by processing the results stored in the evaluation vector and in the matrices of discriminators.



Capture 1. The *OPTCHOICE* software for MADM modeling and solving optimal choice problems

In order to understand how *OPTCHOICE* works, one starts with its hardware and software platform presentation. Like all advanced software working on the Internet, *OPTCHOICE* has two functional blocks. The first block, installed on a powerful MySQL, MAIL and WEB server, is written in PHP and CLIPS, and addresses registration in the system and models building as human jobs. The second block, installed on a variable number, from 4 to 10, of Symmetric Multi-Processors (SMP), is written in C++ and OpenMP [12], and addresses generating and solving problems as automatic jobs. Both blocks function under WINDOWS 2003. Installed on the server, an automation program, whose role is to exclude the human intervention in the computing and communication processes, links these two blocks.

After the procedure of logging into the system is complete, one can proceed to build MADM models. The log-in procedure is not overly restrictive as, by design, its role is not to discourage using *OPTCHOICE*. Building and validating MADM models involve tens of users working simultaneously with *OPTCHOICE* on the Internet. This type of parallelism, which is named *concurrency on server*, is not dis-

cussed in this paper. However, one mentions that the maximum number of users who can work concurrently on server is 100. This is a limitation generated by the type of server.

If 100 *OPTCHOICE* users are dealing over the Internet, at the same time, with problems involving 10 decision-makers, 5 states of nature, 250 objects and 50 attributes, which are considered medium-size problems by most people, it is obvious that, without an efficient technical platform and an effective parallel computing technique, the main characteristic of *OPTCHOICE,* namely pervasiveness, is compromised.

2 Load Balancing Algorithm for the Set of OCPs Launched from the Internet

Subsequent to the installation of the software modules on the computing equipment, the MADM database's modules on the MySQL, WEB and MAIL server, and the optimization modules on the multi-processors hosting the OCPs solving, the automation program must be lunched. This program, named BF1-BF2_Scheduler, is designed to read the identification coordinates of the problems to be solved, launch the solving processes, manage the executions optimally and track the completions of the executions. From the point of view of the connection with the first computing system, the scheduler only acts on the entity WAITING_LINE, which contains only the codes of the problems and the status of the associated solving processes; from the point of view of the second computing system, the scheduler only uses its internal memory for processes launching.

Basically, the program executes an infinite cycle which consists of sequential steps, iterated always from the first step. If idle for an established time, the scheduler becomes dormant until a waking event is received from the first computer. Such an event occurs when the coordinate of a new problem appears in the waiting queue. The parameters of the scheduler, namely *the elapsed time since the last ending of a solving procedure,* after which the scheduler goes dormant, and *the maximum time allocated to a solving procedure* are handled by the system administrator who fixes them in the memory.

Step 0: The dormant scheduler wakes up if a waking event is received from the server;

Step 1: Determine the first problem to be solved (with *Status*=0) in the waiting queue;

Step 2: Delete from the waiting queue all the duplications of this problem to prevent multiple executions on the same data set of the problem;

Step 3: Launch the execution of the problem determined at Step 1 on the multi-processor with the easiest load (mod *Status*=1);

Step 4: Delete from the waiting queue all the problems whose executions have been successfully completed (*Status*=2, status modified at the end of each problem solving process by the hosting multi-processor);

Step 5: For each problem in execution (with *Status*=1), measure the elapsed time since launching its execution and if this time exceeds the maximum admissible value, create conditions for stopping, check the running troubles, and re-launch its execution;

Step 6: Measure the time elapsed since the last operation executed to the benefit of the *OPTCHOICE* system and if it exceeds the maximum admissible value, then command the scheduler to return to *Step 1* and, if no problems are identified, go in dormant state.

This algorithm solve the problem of uniform computing distribution over the set of multi-processors hosting the OCPs solving but do not any time reduction on a multi-processor level. For this purpose it is necessary to exploit the parallelism facility of each multi-processor. In the folowing will be shown how this thing is possible.

3 MADM General Models Building and the *OPTCHOICE* Relational Database

It is well known that every parallel computing has at its base an analysis on data structures and an analysis on algorithms' structure. In this section the analysis on data will be done.

3.1 MADM General Models Building

Building a MADM model in a given domain requires establishing the decisional context in which the optimal object o^* will be selected from a set $O = \{ o[i] \mid i = \overline{1,\mathbf{i}} \}$ of objects. As a first step in the process of building a model, the manager of the corresponding domain establishes the set of decision makers $D = \{ d[l] \mid l = \overline{1,\mathbf{l}} \}$, i.e. the persons who will have responsibilities in the process of building and validating the model, as well as in generating and solving OCPs. Typically, the decision makers discuss and aggree on what their absolute weights $W\_D = \{ w\_d[l] \mid l = \overline{1,\mathbf{l}} \}$ will be, with $\sum_{l=1}^{\mathbf{l}} w\_d[l] = 1$. However, if a consensus is not reached, then all the decision makers provide their own vectors of weights and *OPTCHOICE* will calculate automatically the absolute weight of each decision maker.

The first task of the decision makers is to establish, independent of each other, the set of states of nature $S = \{ s[k] \mid k = \overline{1,\mathbf{k}} \}$. A state of nature is defined as the totality of conditions defined on the given domain which determine, for the objects taken into consideration, certain values of attributes. It is clear that when a new state of nature is entered into the

system by a decision maker, the other decision makers learn about it and contribute to its good definition. The absolute weights of the states of nature, $W\_S = \{ w\_s[k] \mid k = \overline{1,\mathbf{k}} \}$, with $\sum_{k=1}^{\mathbf{k}} w\_s[k] = 1$, are determined through a process similar to the process of determining the absolute weights of the decision makers, i.e. either directly, by consensus, or by using *OPTCHOICE*.

The second task of the decision makers is to identify the set of attributes $A = \{ a[j] \mid j = \overline{1,\mathbf{j}} \}$. The attributes are characteristics of the objects in terms of which they are evaluated in order to determine the optimal object. This set is obtained as the union of the subsets of attributes specified by each decision maker. The absolute weights of the attributes, $W\_A = \{ w\_a[j] \mid j = \overline{1,\mathbf{j}} \}$, with $\sum_{j=1}^{\mathbf{j}} w\_a[j] = 1$, are determined according to the recipe already described for decision makers and states of nature.

For each attribute, one gives its interval of variation ($lo\_a[j]$, $up\_a[j]$) and the optimization sense i.e. *min* or *max*. Eventually, the decision makers must enter the components $lo\_a\ [j] \le c_{lkij} \le up\_a\ [j]$ of the matrices OA_{lk}, for each element of the cartesian product D X S. The generic element c_{lkij} represents the value of the atribute j of the object i corresponding to the state of nature k and determined by the decision maker l. The four-dimensional array obtained in this way is reduced to ***l*x*k*** two-dimensional arrays (matrices) for a couple of reasons: first, most people are familiar with matrix manipulations, and second, the methods of solving problems generated by reduced to one decision maker and one state of nature MADM models are based on two-dimensional model. Initially, these matrices have a hybrid character, being divided, intuitively speaking, into two areas: the *well-defined area*, in which every attribute has well-defined values for every object, for every state of nature, and in the opinion of every decision maker, and the *ill-defined area*, in which the values of certain attributes in relation to certain objects, in certain states of nature, are either unknown or they cannot be determined by some of the decision makers, possibly by most of them. One can speak of a well-defined area and an ill-defined area because, as seen above, it is up to the human factor to provide the matrix entries. Incompleteness is an error easy to detect in the model. But the human factor is also at the base of other types of errors, more insidious, regarding incorrectness and incredibleness. Syntactic or semantic errors appear infrequently, due to strong validation procedures incorporated in *OPTCHOICE*. The productions set P (expressed in the general format *IF* $cond_1 \Lambda\ cond_2 \Lambda \ldots \Lambda\ cond_m$ *THEN* $act_1, act_2, \ldots, act_n$), created by independent experts and working on model data, is used to remove any kind of inconsistency.

3.2 MADM General Models' Database

The projection of the MADM model, generator of OCPs, on data structures organized and managed with an SGBD is called an *OPTCHOICE database*. Relational databases are the most frequently used type of databases in mathematical modeling and in this case are the only recommended databases.

The *OPTCHOICE* database contains the main entities and the link entities. While the main entities describe specific objects of the model (decision makers, states of nature, objects, attributes, problems etc.), the link entities describe relations between two real objects (decision makers – states of nature, states of nature – objects, objects – attributes, problems – decision makers etc.). It is to notice that the database contains not only fields corresponding to the MADM models but also fields that correspond to the *OPTCHOICE* problems to be generated on the base of existing data.

The ensemble of the database entities and relations is contained in the following relational diagram:

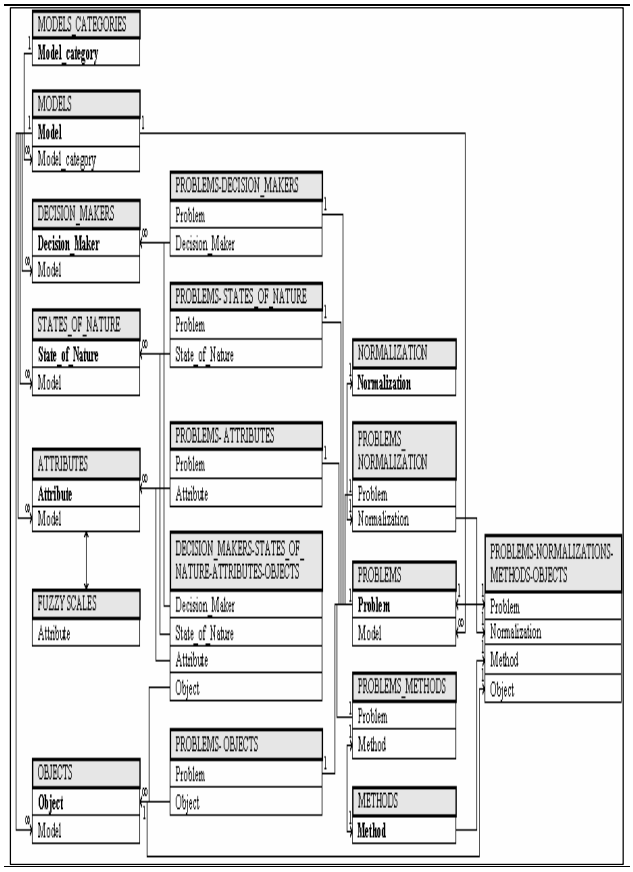


Diagram 1. Structure of the *OPTCHOICE* database

It is obvious that an analysis on *OPTCHOICE* database brings the first ideas about the feasibility of a parallel algorithm that must be conceived later. More precisely, the hierarchical structure, observed in the above diagram, make possible to separate data with the same structure and significance as entries for independent

repeating processes. This is a valuable hint for continuing the analysis.

4 Parallel Computing for an OCP

When generating OCPs, the input is a consistent MADM model in the database together with a set of instancing parameters, and the output consists of a set of scalars, vectors and arrays structured in a way that facilitates solving the problem by parallel computing. For a problem, the input parameters add to its name the decision makers, the states of nature, the attributes, the objects and the solving methods involved. OCPs that involve multiple decision makers and multiple states of nature represent natural extensions of single decision maker / single state of nature problems. While in the past different approaches for these extensions have been considered, in this paper the problems are approached unitarily in all their instances: with multiple decision makers and multiple states of nature, with multiples decision makers and a single state of nature, with a single decision maker and multiple states of nature, and with a single decision maker and a single state of nature. When some problems are concurrently generated, their solving is automatically triggered. After solving, the solutions are stored in the *OPTCHOICE* database and so, they are available to the users. It is necessary that, for each problem, the time between the moment when a solving process is launched and the moment of the query for displaying the solution exceeds the solving time. Therefore the waiting for solution is excluded.

4.1 Parallel algorithm's description

As shown before, a repetitive multi-level structure in the MADM models facilitates parallel computing. The decisional framework for evaluating objects is defined hierarchically, decision makers – states of nature – attributes, which naturally yields the possibility of defining a method of decomposition [13], [14].

Solving a general OCP with multiple decision makers and multiple states of nature involves the construction of a special routed tree. The root of this tree, the 0 level, is represented by the well-defined model with an instance, i.e. the generated problem; at level 1 in this tree are the decision makers $d_1, d_2, \ldots, d_{\mathbf{l}}$; at level 2 are the states of nature $s_1, s_2, \ldots, s_{\mathbf{k}}$; finally, the at terminal level, level 3, are the attributes $a_1, a_2, \ldots, a_{\mathbf{j}}$. Note the main property of this routed tree is that, for a fixed level, every node situated on this level has the same children as number and significance. This claim is supported by the fact that if at the begining each decision maker from the set D establishes, independent of the other decision makers, the sets of states of nature and attributes that he/she finds relevant for modeling, eventually all decision makers aggree on the same set of states of nature S and the same set of attributes A, each decision maker being expected to assign values to all the attributes and in all the states of nature considered in this tree. An element at the leafs' level of the tree, denoted in the figure by C_{ilkj} (where $i=\overline{1,\mathbf{i}}$, $l=\overline{1,\mathbf{l}}$, $k=\overline{1,\mathbf{k}}$, $j=\overline{1,\mathbf{j}}$), in which the last three subscripts follow the hierarchy defined in the tree, has the following significance: C_{ilkj} is the value of the attribute $a[j]$ of the object $o[i]$, in the state of nature $s[k]$, given by the decision maker $d[l]$. In addition, $w\_d[1]$, $w\_d[2]$, ..., $w\_d[\mathbf{l}]$, - the weights of decision makers, $w\_s[1]$, $w\_s[2]$, ..., $w\_s[\mathbf{k}]$ - the weights of states of nature and $w\_a[1]$, $w\_a[2]$, ..., $w\_a[\mathbf{j}]$ - the weights of attributes are arranged on the tree levels according to the established hierarchy.

Note that if, for instance, $d[1]$ and $s[1]$ are fixed, then one obtains a problem which is solved by using only the entities $O = \{ o[i] \mid i=\overline{1,\mathbf{i}} \}$ and $A = \{ a[j] \mid j=\overline{1,\mathbf{j}} \}$, and the set of attributes weights $W\_A = \{ w\_a[j] \mid j=\overline{1,\mathbf{j}} \}$, i.e. a classical single decision maker and single state of nature OCP. This problem represents a part of the array depicted in Figure 1.

Just as this problem was built, if one considers the cartesian product of the sets of decision makers and states of nature, then one can construct **lxk** two-dimensional problems of size (**ixj**) which can be solved separately but in parallel (see again Figure 1), reducing this level of the graph. Solving these problems yields solutions which are stored into the array $\{ C_{ilk} \}$ (where $i=\overline{1,\mathbf{i}}$, $l=\overline{1,\mathbf{l}}$, $k=\overline{1,\mathbf{k}}$), in which the last two subscripts are according to the established hierarchy. This array is transferred to the reduced tree at the states of nature level, which, in this way, becomes the new terminal level. Similar to the previous level, one also takes into consideration the weights of the states of nature $W\_S = \{ w\_s[k] \mid k=\overline{1,\mathbf{k}} \}$(see Figure 2).

In order to preserve the methodological coherence, as done at the previous step, one continues by solving in parallel **l** two-dimensional problems of size (**ixk**) and the tree is reduced again by one level. The solutions, which are stored in the array $\{ C_{il} \}$ (where $i=\overline{1,\mathbf{i}}$, $l=\overline{1,\mathbf{l}}$), are transferred to the reduced tree at the decision makers level along with the decision makers weights $W\_D = \{ w\_d[l] \mid l=\overline{1,\mathbf{l}} \}$(see Figure 3). In this way, one last two-dimensional problem of size (**ixl**) needs to be solved. Solving this problem produces a solution $\{ C_i \}$ (where $i=\overline{1,\mathbf{i}}$) (see Figure 4, and the corresponding final values called *merits*).

This decomposition method is very prodigious because it naturaly generates all the benefits of treating OCPs by parallel computing. Using the same data structures and dimensions and, for a fixed solving

method, the same algoritm in all the nodes and on all the levels, are elements that improve the time perfomance in parallel computing because the parallel processes take aproximately the same amount of time and, as a consequence, the waiting time between processes is insignificant. On one level, the generating of sub-problems and their launching in execution are made from left to right.

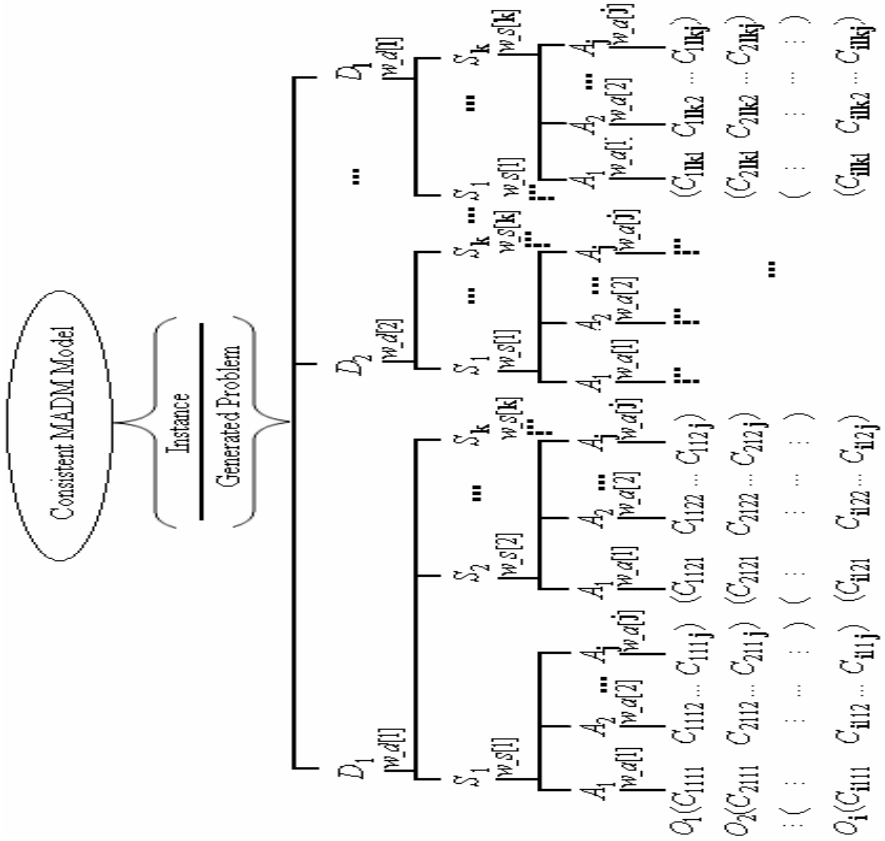


Figure 1. Attributes' Level Processing

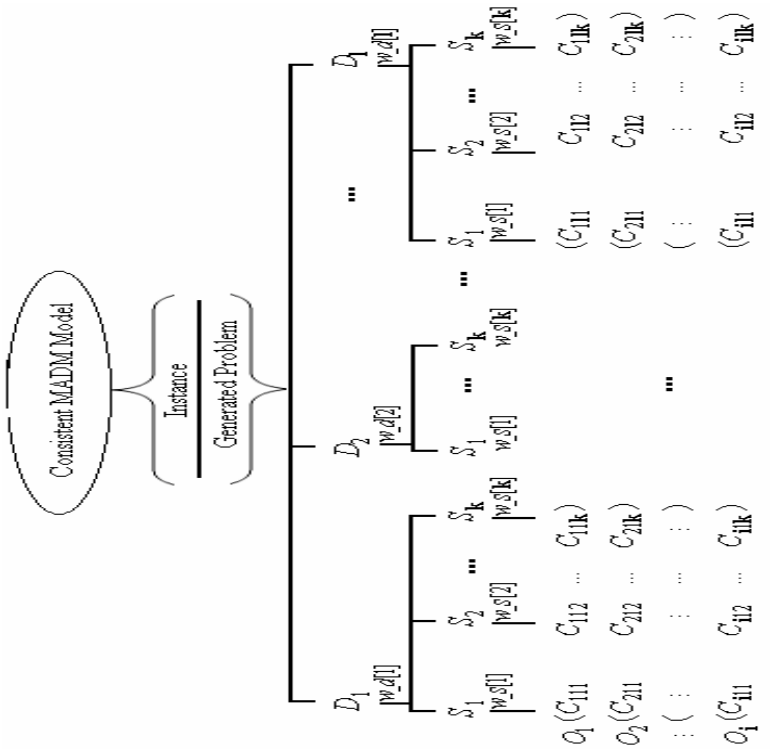


Figure 2. States of Nature's Level Processing

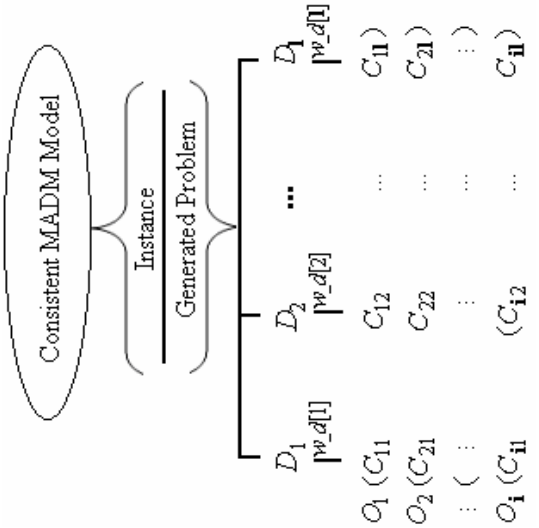


Figure 3. Decision Makers' Level Processing

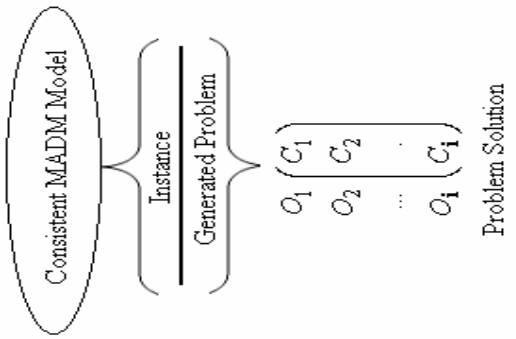


Figure 4. Problem's Optimal Solution

Remarks:

1. At each level of solving sub-problems, one needs to take into account the weights of the main entities (attributes, states of nature, and decision makers); if these weights are not present in the problem data, they are set equal with each other;

2. The intervals of variation and the optimization sense (minimum or maximum) of the attributes are normalized so that each attribute variation interval becomes [0, 1] and each optimization sense becomes *maximum*. This ensures that the problems are in the same class at each level of solving;

3. Following the dimension reduction given by attributes, there are two alternative approaches to continue solving the initial problem: a first possibility is to aggregate the results corresponding to each state of nature taken individually, for all the decision makers; a second possibility is to aggregate the results for all the states of nature, for each individual decision maker. Since the approach is based on the principle of solving the initial problem from bottom to top and from left to right (as it is usual in graph theory which in this case is in resonance with decision theory), the later alternative is preferred, so that the solving method is concordant with the problem decomposition idea;

4. If one of the levels, namely levels 1 or 2, is reduced to a single node, the procedure is the same, the two-dimensional algorithm being capable of approaching any data configuration;

5. With the above comments, the algorithm is coherent and can be used in two different instances, both decisionally correct: a) from bottom to top, where the same MADM solving method is used on each level, and b) different MADM solving methods are used on different levels, if the decision makers find this appropriate.

Solving a single OCP problem involves a sequence of three distinct operations. The first operation is to generate the problem in the memory of the computer chosen for hosting the solving processes, starting from the coordinates of the problem from the waiting queue and from the data stored in the database. The data configuration of the problem in the internal memory of the solver's host differs significantly from the data configuration in the database. This difference is normal, as the data are organized in the external memory in a way that allows optimal data storing / retrieving, while in the internal memory the data are organized in a way that facilitates executions of algorithms and computations. In addition, in this case

the model's data are transformed into the problem's data by applying one normalization method. The methods of normalization are mathematical methods that perform an initial processing of the model's data. This operation has no significance for the user, but it is required by most solving methods of the OCPs. A function, usually linear, is used for each attribute such that the least favorable value of the attribute $lo\_a[j]$ maps to the normalized value 0, whereas the optimal value of the attribute $up\_a[j]$ maps to the normalized value 1. In this way, all the attributes are transformed to have a maximal optimization sense.

The internal memory of the OCPs solver's host computer will contain, at this stage, the data necessary and sufficient for solving the problem according to its definition:

- WProblem, *- MM*, *WMethod(MM), WMWeight(MM),* *- DD*, *WDecident(DD)*, *WDWeight(DD),* *- SS, WState(SS)*, *WSWeight(SS),* *- AA*, *WAttribute(AA)*, *WAWeight(AA),* *- OO, WObject(OO)*, *WEval(OO, MM),* *- WValue(DD, SS, AA, OO).*
- Problem code, - For the decisional context: # of entities, entities, weight of entities, - For the decisional variables: # of objects, objects, objects' evaluation, - For the characteristics' array: normalized values.

The second operation is to solve the problem by using the method selected from the set of available solving methods. It is well known that the multi-processor is of the SMP-type and so, the method work on the data block described above. Assigning to *PP*, the cardinal of the processors' set, for each level in the described routed tree, one can solve in parallel *PP* two-dimensional problems. Repeating the procedure for all levels, the problem is solved for a given method. In order to illustrate what a processor does for a certain solving method, a well-known method, in the format of a procedure / function / subroutine / future thread, written in a pseudo-code, is presented next. Its simplicity assures a minimum running time.

The TOPSIS Method

Since, by normalization, the optimization sense is maximal for all the attributes, the ideal point is the vector of dimension *CC* with all the components equal to 1. The least preferable point is the vector of dimension *CC* that has all the components equal to 0. For each object *O*, with *O*=1,*OO*, the geometric distances to these two points are calculated by taking the square roots of the sums (over *C*) of the squared differences *(*1- *WNValue(C,O))***2 and *(*0-*WNValue(C,O))***2, respectively. These distances, weighted by *WNWeight(C),* are used to construct directly a merit of each object, which is stored in the vector *WNValue(*0, *OO).*

```
PROCEDURE_BF2_TOPSIS (CC, OO,
                      WNValue(CC, OO),
                      WNWeight(CC))
BEGIN PROCEDURE
INTEGER C, O
REAL WDistancePlus, WDistanceMinus,
WNValue(CC, OO), WNWeight(CC)
DO FOR O=1,OO
WDistancePlus=0
WDistanceMinus=0
DO C=1,CC
   WDistancePlus= WDistancePlus +
     + WNWeight(C)*(1-WNValue(C, O))**2
   WDistanceMinus= WDistanceMinus +
     + WNWeight(C)*WNValue(C, O)**2
ENDDO
WNValue(0,O)= WDistanceMinus / (WDistanceMinus
+ WDistancePlus)
ENDDO
ENDPROCEDURE
```

The parallel algorithm

```
PROCEDURE_BF2_PARALLEL_SOLVING_OF_
AN_OCP (WProblem,
        DD, WDecident(DD), WDWeight(DD),
        SS, WState(SS), WSWeight(SS),
        AA, WAttribute(AA), WAWeight(AA),
        OO, WObject(OO), WEval(OO),
        WValue(DD, SS, AA, OO),
        CC, OO, NValue(CC, OO),
        WNWeight(CC))
BEGIN PROCEDURE
INTEGER DD, SS, AA, CC, OO, PP
        WDecident(DD),
        WState(SS),
        WAttribute(AA),
        WObject(OO)
REAL    WDWeight(DD),
        WSWeight(SS),
        WAWeight(AA),
        WEval(OO),
        WValue(DD, SS, AA, OO)
READ FROM OPTCHOICE DATABASE:
WProblem,
DD, WDecident(DD), WDWeight(DD),
 SS, WState(SS), WSWeight(SS),
AA, WAttribute(AA), WAWeight(AA),
OO, WObject(OO), WEval(OO),
WValue(DD, SS, AA, OO),
CC, OO, NValue(CC, OO),
WNWeight(CC)
```

BUILD *WNValue(AA, OO)* FROM *WNValue(DD, SS, AA, OO)*
DO IN PP - PARALLEL FOR *A*=1,*DD*SS*
PROCEDURE_BF2_TOPSIS*(AA, OO, WNValue(AA, OO), WAWeight(AA))*
WAIT FOR LAST PROCESS
ENDDO
BUILD *WNValue(SS, OO)* FROM *WNValue(AA, OO)*
DO IN PP - PARALLEL FOR *D*=1,*DD*
PROCEDURE_BF2_TOPSIS*(SS, OO, WNValue(SS, OO), WAWeight(SS))*
WAIT FOR LAST PROCESS
ENDDO
BUILD *WNValue(DD, OO)* FROM *WNValue(SS, OO)*
DO PROCEDURE_BF2_TOPSIS*(DD, OO, WNValue(DD, OO), WAWeight(DD))*
ENDDO
BUILD *WNValue(OO)* FROM *WNValue(DD, OO)*
UPDATE *OPTCHOICE DATABASE* WITH *WNValue(OO, TOPSIS)* ASSOCIATED TO *WProblem*
ENDPROCEDURE

For a single OCP it is recommended to run more then one solving method. Sometimes the solutions can be different. This conducts to a decisional dilemma. *OPTCHOICE* must have a procedure for finding the global optimum as the final step in solving the OCP. Using the optimal solutions given by the mathematical methods used, which are stored in the matrix *WEval(OO, MM)*, the weighted average of the columns of this matrix is calculated, with the weights stored in the vector of method weights, and the amended result (by the information from the matrix of discriminators) is stored in the column vector *WEval(OO,* 0*)* as the global optimum.

After multi-solving the OCP, the optimal solutions are stored in the database. Starting with the data in the multi-processor memory, one accesses the server's database, and its entities *PROBLEMS – NORMALIZATIONS – METHODS - OBJECTS* and *WAITING_LINE* are updated. Throughout the solving process of a problem, the user must be warned that the solving process is being executed, specifying the elapsed time.

5 Conclusions

The natural hierarchical structure of a general MADM model allows solving OCPs, defined on this model, by parallel computing. Conceptually, the mathematical MADM model is represented as a tree in which the levels correspond to main entities (decision makers, states of nature, and attributes). The solving method implemented in *OPTCHOICE* divides the problem into sub-problems at each level of the tree. When all the sub-problems at a terminal level of the tree are solved, the terminal level collapses, and the resulted solutions are fed as data to other sub-problems at the next level, which becomes the new terminal level. While *OPTCHOICE* is flexible enough to allow applying different solving methods at different levels, it is recommended to apply the same solving method systematically, at all levels.

The *OPTCHOICE* software belongs to a couple of integrated instruments for promoting the enhanced using of the MADM domain. Beside this pervasive service for MADM modeling and OCPs generating and solving, a MADM *e-course* is under construction. One of the modules of this e-course is a tutorial on *OPTCHOICE*. Therefore, the designers intend to offer the opportunity to help users to become familiar with the software before using it for real-life problems.

The *OPTCHOICE*'s design and programming consists of 7 analysts and programmers. The work started in 2006 and its completion is scheduled for 2008. The partial results are encouraging, the simulations made for an incomplete set of MADM solving methods, in a static context, showing good results.

References:

[1] http://www-neos.mcs.anl.gov/
[2] http://www.sce.carleton.ca/faculty/chinneck /StudentOR_ /html
[3] http://home.ubalt.edu/ntsbarsh/zero/scientific Cal.htm #rmenu
[4] http://plato.la.asu.edu/guide.html
[5] Cohen M.-d., Kelly, C.B., Medaglia, A.L.: Decision Support Systems with Web-Enabled software. *Interfaces* 31(2), (2001), pp. 109-129.
[6] Hwang, C-L., Yoon, K.: Multiple Attribute Decision Making. Springer-Verlag, Berlin Heidelberg New York (1981).
[7] Hwang, C-L., Lin, M.J.: Group Decision Making under Multiple Criteria. Springer-Verlag, Berlin Heidelberg New York (1997).
[8] Resteanu, C., Filip, F.G., Ionescu, C., Somodi, M.: On Optimal Choice Problem Solving. In *Sage, A.P., Zheng,W., (eds.): Proceedings of SMC '96 Congress (Beijing, October 14-17)*. IEEE Publishing House, Piscataway NJ (1996), pp. 1864–1869.
[9] Dongarra J., Madsen, K., Wasniewski, J. (Eds): Applied Parallel Computing: State of the Art in Scientific Computing. *Lecture Notes in Computer Science,* Springer; 1 edition (April 11, 2006)
[10] http://www.wordreference.com/definition /pervasive
[11] Giarratano, J.C., Riley, G.D.: Expert Systems: Principles and Programming. 3rd edition. PWS Publishing Company, Boston, (1999).
[12] Quinn, M.: Parallel Programming in C with MPI and OpenMP. McGraw-Hill Science / Engineering/Math; 1 edition (2003).
[13] Jordan, H.F., Alaghband, G., Jordan, H.F.: Fundamentals of Parallel Processing. Prentice Hall (2002).
[14] Grama, A., Karpis, G., Kumar, V., Gupta, A.: Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms. Addison Wesley (2003).